# New Generation of UPPAAL *

*Johan Bengtsson*[2] *Kim Larsen*[1] *Fredrik Larsson*[2]
*Paul Pettersson*[2] *Yi Wang* [2] *Carsten Weise*[1]

[1] BRICS, Dept of Computer Science, Aalborg University, Denmark.
[2] Department of Computer Systems, Uppsala University, Sweden.

**Abstract.** UPPAAL is a tool-set for the design and analysis of real-time systems. In [6] a relatively complete description of UPPAAL before 1997 has been given. This paper is focused on the most recent developments and also to complement the paper of [6].

## 1 UPPAAL's Past: the History

The first prototype of UPPAAL, named TAB at the time, was developed at Uppsala University in 1993 by Wang Yi et al. Its theoretical foundation was presented in FORTE94 [11] and the initial design was to check safety properties that can be formalized as simple reachability properties for networks of timed automata. The restriction to this simple class of properties was in sharp contrast to other real-time verification tools at that time, which where developed to check timed bisimularities or formulae of timed modal $\mu$-calculi. However, the ambition of catering for more complicated formulae lead to extremely severe restrictions in the size of systems that could be verified by those tools.

The essential ideas behind TAB were to represent the state space of timed systems by simple constraints and to explore the state space by constraint manipulation. In 1995, Aalborg University joined the development, and shortly after a C++-version with efficient operations on constraints and checks for inclusion between constraints was finalized. TAB was subsequently renamed UPPAAL with UPP standing for Uppsala and AAL for Aalborg.

Since its first release in 1995, UPPAAL has in numerous case-studies proved itself useful in the analysis of safety properties of extremely complicated system descriptions. To our knowledge, UPPAAL has at present over one hundred users in both academia and industry. However, the number of downloads of UPPAAL binary code from the UPPAAL WWW-page is much larger according to the record of our WWW-server.

* UPPAAL is developed in collaboration between the Department of Computer Systems at Uppsala University, Sweden and BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation) at Aalborg University, Denmark. The people involved with the development are Wang Yi, Kim Larsen, Paul Pettersson, Johan Bengtsson, Fredrik Larsson, Kåre J. Kristoffersen, Carsten Weise, Per S. Jensen, and Thomas M. Sørensen.

For a detailed description for UPPAAL before 1997, we refer to [6]. During 1997, UPPAAL has been greatly improved e.g. the verification time for the well known Philips audio protocol [1] is reduced from 304 seconds to 5.5 seconds using the same hardware. In Sections 2 and 3 we report on this evolution and on the most recently added model-checking features such as facilities for checking time-bounded as well as (ordinary) liveness properties.

Right from the beginning UPPAAL has been applied in a number of case studies including an rapidly increasing number of case-studies with industrial collaboration. To meet requirements arising from the case studies, UPPAAL has been extended with various features leading to the current distributed version. In Section 4 we offer a brief summary of recent case-studies undertaken by UPPAAL.

The success of UPPAAL efficiency-wise has lead to a strong demand of a reimplementation of the graphical user interface. In particular, the verification engine of UPPAAL now routinely handles models which are too big to be displayed in full on a single screen: thus the ability to perform editing as well as simulation, while focusing only on a selection of relevant components is highly needed. The present distribution of UPPAAL contains two separated graphical tools: an AUTOGRAPH-based *editor* and a graphical *simulator* implemented in XFORMS and *Motif*. From a users perspective one single graphical interface would be preferable. In Section 5 we report on a new UPPAAL graphical user interface currently under implementation addressing these points.

## 2 UPPAAL's New Languages

Two major improvements have been made on the modeling and specification languages. First, we have introduced two new types: *bounded* integer and *array* of such integers, to simplify modeling. Second, the verifier has been extended to handle liveness properties in addition to reachability properties.

**Bounded integers and arrays of integers** Instead of using a default domain derived from the hardware implementation of integers, we now allow the user to specify the domains of each variable. However, if no domain is given by the user, a default domain (currently $[-32768, 32767]$) will be assigned to the variable. When assigning a value to a variable, the value is "wrapped" into the correct domain.

In order to ease the modeling task, the class of integer expressions handled by the verifier have been extended. As shown in Figure 1, the new verifier can handle general expressions over integer variables and constants. To allow more condensed models, arrays of integers and arithmetic if-statements have been added to the language. The syntax used for there constructs is the same as in

the C programming language, i.e. `var[5]` denotes the sixth element of the array `var` and `(x<=5:2?3)` is 2 if the value of `x` is at most 5, and 3 otherwise.

*IExp* → *Id* | *Id* `[` *IExp* `]` | *Nat* | `-` *IExp* | `(` *IExp* `)` | *IExp Op IExp* | `(` *IRel* `?` *IExp* `:` *IExp* `)`
*IRel* → *IExp RelOp IExp*
*Op* → `+` | `-` | `*` | `/`
*RelOp* → `<=` | `>=` | `==` | `!=` | `<` | `>`

**Fig. 1.** Syntax for integer expressions

**Specifying liveness properties** In addition to the reachability properties checked by the older UPPAAL versions, all versions above 2.12 are also capable of checking simple liveness properties. The liveness properties that can be checked are of the form $\exists\Box P$ and $\forall\Diamond P$, where $P$ is a "local" property of the same kind as the properties handled by the reachability checker.

The actual checking is done by searching for a time divergent path from the initial state, where $P$ holds in all states (in case of a $\exists\Box$ property), or where $\neg P$ holds in all states (in case of a $\forall\Diamond$ property).

**Checking Deadlock-Freedom and Consistency** In addition to the updates in the modeling and specification languages, the new version of UPPAAL also contains some features to simplify debugging. During verification the tool reports all inconsistent states (i.e. states where the location invariant is violated when the location is entered) and all deadlocked states (i.e. states where no discrete transition will be possible in the future) encountered.

## 3 UPPAAL's New Heart

In our previous work, before 1997, we have developed and implemented various techniques for optimizing the space- and time-performance of the reachability engine of UPPAAL [7]. The two major optimizations are an algorithm for compaction of constraints and a control structure analysis technique that identifies and discards states that are not necessary to ensure termination of the reachability algorithm [8]. When combined the two techniques yield significant space savings[1] and (usually) improved time-performance.

During 1997, a large part of the source code of the UPPAAL model checker was rewritten and optimized. Surprisingly, small obvious improvements on the

[1] The space saving on the examples in [8] are between 75% and 94%.

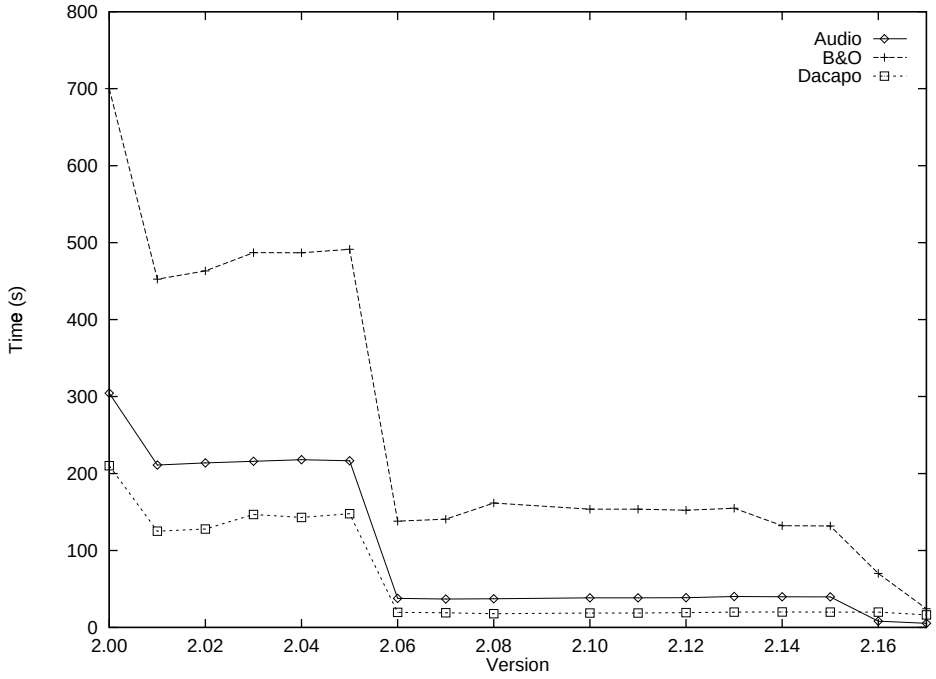


**Fig. 2.** Time benchmark for UPPAAL version 2.00–2.17.

source code, often yield huge improvements in efficiency. The most widely distributed version of UPPAAL is version 2.02, which is also the version presented in the paper [6]. However, the most efficient version is the current one 2.17.

In Figure 2 and 3, we illustrate the improvement of time and memory usage of UPPAAL from version 2.00[2] to 2.17[3] in terms of three case studies; the Philips audio control protocol with bus collision [1], the B&O protocol [5] and the Dacpo protocol [10]. All versions of UPPAAL used in the test were compiled using GCC version 2.7.2.3 and the benchmark was made on a Pentium-II/333 system with 128 MB of main memory, running RedHat Linux 5.0.

In particular, we notice that for both of the time and space usage diagrams, there is a breaking point in version 2.06 compared with the proceeding version. This is due to a number of of internal improvements in the source code including reimplementation of the main data structure i.e. the passed-list.

In the following, we mention a few recent improvements in the implementation.

[2] Version 2.00 is dated Feb 1997.
[3] Version 2.17 was released in March 1998.

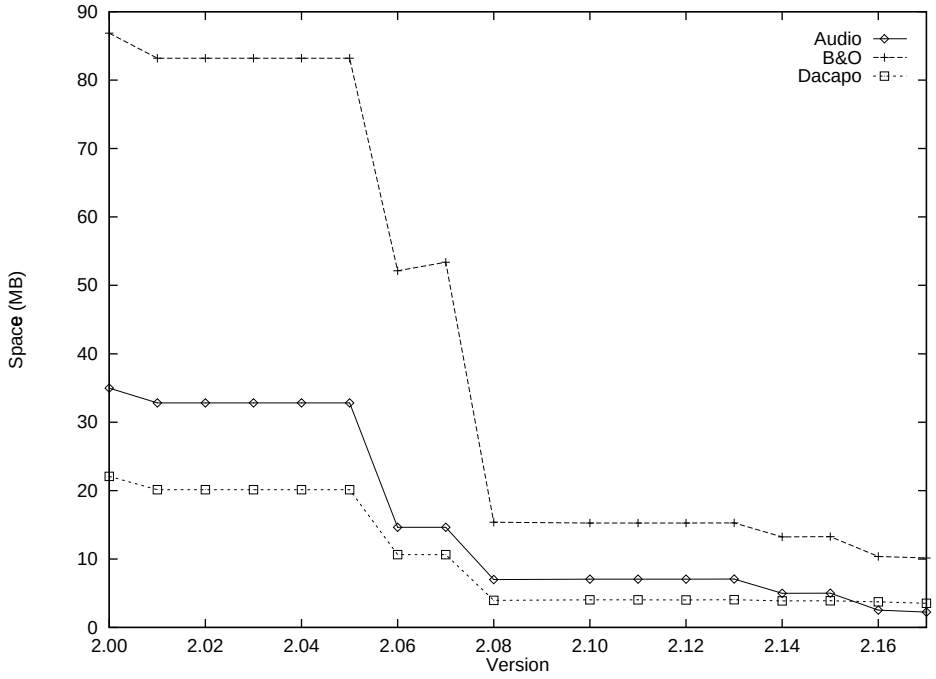


**Fig. 3.** Space benchmark for UPPAAL version 2.00–2.17.

**Improved hash function** The most critical data structure in UPPAAL is the so called passed list. It holds all symbolic states visited during the state space exploration. It is mainly to guarantee termination and to avoid repeated searching. It is often the case that a large portion of time usage is spent on searching through the list.

The passed list is implemented as a hash table with symbolic states as entries. In the previous version of the verifier, the hashing was done exclusively on the control nodes of the automata. Now the hash function also takes the values of data variables into account. The hash function assigns a unique integer to every combination of control nodes and the current values of data variables. This is possible because the number of control nodes is finite and all variables have a given domain i.e. a finite number of different values. This integer can be very large, much larger than the size of the hash table, which means that collisions can still occur even though the integer is unique for each combination.

**Optimized constraint manipulation** UPPAAL represents the symbolic states of a real-time system as constraints over clocks. To keep the constraint manipulation efficient, UPPAAL transforms every constraint system to a canonical form.

This transformation is the most time-consuming of all the operations on the constraints.

There is no way to check if a given constraint system already has this canonical representation which is less expensive as the transformation itself. In the previous version of UPPAAL it happened quite often that the transformation was unnecessary, but in the new version no constraint system already on the canonical form is transformed. This leads to better performance, and is one of the explanations of the big performance leap between version 2.05 and 2.06.

## 4 UPPAAL's New Applications

UPPAAL is frequently being applied in various case-studies, both in industry and academia. The two main application areas are real-time controllers and real-time protocols, and the purpose is often to model and analyze existing systems. However, the tool has also been applied to support design and analysis of systems under development. In particular, it has been used to support the design and synthesis of a gear controller that will operate in a modern vehicle. In the following we summarize this and some other recent applications of UPPAAL.

Recently H. Bowman et. al. applied UPPAAL to model and automatically verify an existing lip synchronization algorithm [2]. Such algorithms are used to synchronize multiple information streams sent over a communication network, in this case, audio and video streams of a multimedia application. The previously published algorithm specification is modeled and verified in UPPAAL. Interestingly, the verification reveals some errors in the synchronize algorithm, e.g. that deadlock situations may occur before pre-described error states are reached after an error.

Another application of UPPAAL in the context of audio/video protocols is reported in [5]. In this industrial application, UPPAAL is used to model and prove the correctness of a protocol developed by Bang & Olufsen. The protocol, which is highly dependent on real-time, is used to transmit messages between audio/video components over a single bus. Though it was known to be faulty, the error was not found using conventional testing methods. Using UPPAAL, an error-trace is automatically produced which revealed the error, furthermore, a correction is suggested and automatically proved using UPPAAL.

D'Argenio et. al. applied UPPAAL to the bounded retransmission protocol protocol [3, 4]. The protocol was proposed and studied at *COST 247, International Workshop on Applied Formal Methods in System Design*. It is based on the alternating bit protocol, but allows for a bounded number of retransmissions, as it is intended for use over lossy communication channels. It is reported that a number of properties of the protocol were automatically checked with UPPAAL. In particular, it is shown that the correctness of the protocol is dependent on correctly chosen time-out values.

In [10] UPPAAL is used to formally verify the so-called Dacapo protocol, a time division multiple access (TDMA) based protocol intended for local area networks that operate in modern vehicles. The study focused on analyzing the start-up of the protocol and on deriving an upper-time bound for the start-up to complete. It is proved that a network consisting of three or four nodes is guaranteed to eventually become operational and the upper time-bounds for the start-up to complete is also synthesized. Further, the start-up is shown to eventually complete for networks with a clock drift corresponding to 1/10000 between the nodes.

Another application also within the automotive industry is described in [9]. Here UPPAAL is applied to support the *development* of a system, rather than to analyze an *existing* system. The system is a prototype gear controller developed at Mecel AB in a collaboration project with the Department of Computer Systems at Uppsala University. The gear controller implements a gear change algorithm in the control system of a modern vehicle. It is designed to operate in a given surrounding environment and to satisfy a number of informal requirements prescribed by the engineers at Mecel AB. In the development, the simulator of UPPAAL was frequently used to validate the behavior of the intermediate controller descriptions. The final description was verified to satisfy 46 logical properties derived from the informally prescribed requirements.

## 5 UPPAAL's New Look

Apart from efficiency, the graphical user interface of UPPAAL, which allows easy editing of specifications and visualization of simulation runs, is one of the strong points of UPPAAL. In an upcoming major revision, the graphical interface will be substantially strengthened. This comes together which an extension of UPPAAL's input format, which will help to ease the job of modeling complex systems.

The current distribution of UPPAAL (see [6]) consists of several programs like `checkta` (the syntax checker) and `verifyta` (the modelchecker) which constitute UPPAAL's *engine*, i.e. the algorithmic side of UPPAAL. Further `xuppaal` is a graphical interface using XFORMS, which calls the different programs for the verification and has a built-in graphical interface for visualization of simulation runs as well as an editor for the requirements specification. AUTOGRAPH is used as a graphical editor for UPPAAL specifications, and a special program called `atg2ta` is needed to translate AUTOGRAPH's generic format into UPPAAL's more convenient `.ta` format.

A major disadvantage of this approach is that the look-and-feel of AUTOGRAPH as the graphical editor differs widely from the visualization in the simulator. Therefore UPPAAL98 (see Fig. 4) will have a completely re-designed

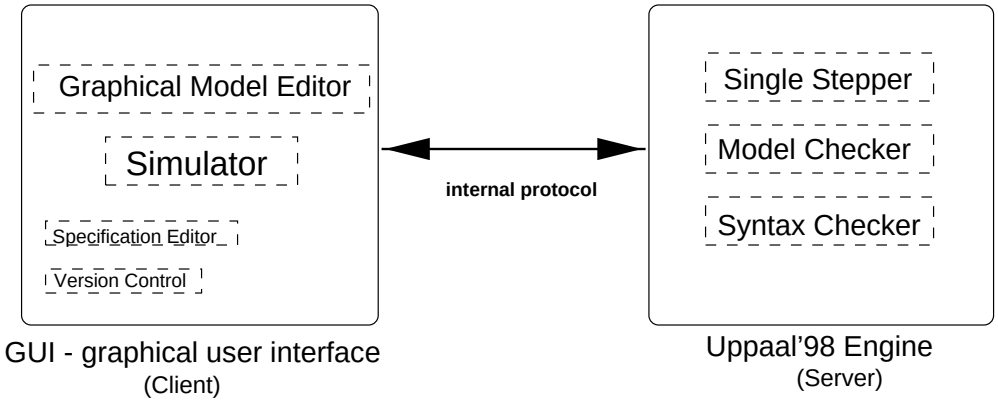


**Fig. 4.** UPPAAL98.

graphical user interface (GUI) unifying the graphical editor and the simulator. This new version is built as a server/client architecture, with UPPAAL's engine as the server and the GUI as the client. As integral parts of the new GUI, the graphical editor and the simulator share the same look-and-feel, which is mainly inspired by the current comfortable, easy-to-use version of `xuppaal`. Additionally the GUI will also include a specification editor and support for version control and documentation of the models and specifications. The heart of the server, which includes the syntax and the model checker, is a *single stepper* which allows to step through the reachability graph of a system. The single stepper is heavily used by the GUI's simulator. In addition to these improvements, the new approach also solves some inconsistencies between the three parts of UPPAAL's current distribution, which lead to problems in the maintenance and even in the usage.

The new GUI is written in JAVA$^{TM}$, making it available for all major platforms. The client/server architecture allows UPPAAL98 to be run either completely locally, client and server residing on the same machine, or to use the graphical interface and the Internet to access a host running the server. By this UPPAAL98 can be directly used via the world wide web, and it especially can be used from platforms on which an executable for the server is not available.

The new GUI also extends UPPAAL's modeling language, so that generic processes can be modeled in order to ease re-usability. The new extended format of UPPAAL's language is downward compatible with the current `.ta` format, so that existing examples will still work with UPPAAL98. The graphical information needed by the graphical editor and the simulator are now stored in a new format internal to the new GUI, so that the `.atg` files are no longer be needed. A translator from `.atg` to the new format will be available for downward compatibility.

A major feature of the simulator is the possibility to blind out parts of the system, so that in a simulation of a large system the user can concentrate on the parts he is really interested in.

At the time being, an internal version of the new GUI is up and running, which is implemented in a generic way, using design patterns from object oriented programming. This makes the GUI flexible to changes and future extensions. This version has been implemented by Carsten Lindholst and Peter Lindstrøm, two computer science students, and Carsten Weise. The server side has been implemented by Fredrik Larsson. A public version of UPPAAL98 is anticipated to be available in July.

## References


1. Johan Bengtsson, David Griffioen, Kåre Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an Audio Protocol with Bus Collision Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of 9th Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 244–256. Springer–Verlag, July 1996.
2. H. Bowman, G. Faconti, J.-P. Katoen, D. Latella, and M. Massink. Automatic Verification of a Lip Synchronisation Algorithm using UPPAAL. In *In Proc. of the 3rd International Workshop on Formal Methods for Industrial Critical Systems*, 1998.
3. P.R. D'Argenio, J.-P., Katoen, T. Ruys, and J. Tretmans. Modeling and Verifying a Bounded Retransmission Protocol. In *Proc. of COST 247, International Workshop on Applied Formal Methods in System Design*, 1996. Also available as Technical Report CTIT 96-22, University of Twente, July 1996.
4. P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In *Proc. of the 3rd Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1217 in Lecture Notes in Computer Science, pages 416–431. Springer–Verlag, April 1997.
5. Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, December 1997.
6. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1997.
7. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL: Status and developments. In Orna Grumberg, editor, *Proc. of 10th Int. Conf. on Computer Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 456–459. Springer–Verlag, June 1997.
8. Fredrik Larsson, Kim G. Larsen, Paul Pettersson, and Wang Yi. Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pages 14–24, December 1997.
9. Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear-Box Controller. In *Proc. of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, March 1998.
10. Henrik Lönn and Paul Pettersson. Formal Verification of a TDMA Protocol Startup Mechanism. In *Proc. of the Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 235–242, December 1997.
11. Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the 7th International Conference on Formal Description Techniques*, 1994.